autogluon_each_location

October 28, 2023

1 Config

[1]:
```
# config

label = 'y'
metric = 'mean_absolute_error'
time_limit = 60*10
presets = "best_quality"#'best_quality'

do_drop_ds = True
# hour, dayofweek, dayofmonth, month, year
use_dt_attrs = []#["hour", "year"]
use_estimated_diff_attr = False
use_is_estimated_attr = True

drop_night_outliers = True
drop_null_outliers = False

# to_drop = ["snow_drift:idx", "snow_density:kgm3", "wind_speed_w_1000hPa:ms",␣
↪"dew_or_rime:idx", "prob_rime:p", "fresh_snow_12h:cm", "fresh_snow_24h:cm",␣
↪"wind_speed_u_10m:ms", "wind_speed_v_10m:ms", "snow_melt_10min:mm",␣
↪"rain_water:kgm2", "dew_point_2m:K", "precip_5min:mm", "absolute_humidity_2m:
↪gm3", "air_density_2m:kgm3"]#, "msl_pressure:hPa", "pressure_50m:hPa", ␣
↪"pressure_100m:hPa"]
to_drop = ["wind_speed_w_1000hPa:ms", "wind_speed_u_10m:ms", "wind_speed_v_10m:
↪ms"]


use_groups = False
n_groups = 8

# auto_stack = True
num_stack_levels = 0
num_bag_folds = None# 8
num_bag_sets = None#20

use_tune_data = True
```

```
use_test_data = True
#tune_and_test_length = 0.5 # 3 months from end
# holdout_frac = None
use_bag_holdout = True # Enable this if there is a large gap between score_val␣
↪and score_test in stack models.

sample_weight = None#'sample_weight' #None
weight_evaluation = False#
sample_weight_estimated = 1
sample_weight_may_july = 1

run_analysis = False

shift_predictions_by_average_of_negatives_then_clip = False
clip_predictions = True
shift_predictions = False
```

2 Loading and preprocessing

[2]:
```
import pandas as pd
import numpy as np



import warnings
warnings.filterwarnings("ignore")


def feature_engineering(X):
    # shift all columns with "1h" in them by 1 hour, so that for index 16:00,␣
↪we have the values from 17:00
    # but only for the columns with "1h" in the name
    #X_shifted = X.filter(regex="\dh").shift(-1, axis=1)
    #print(f"Number of columns with 1h in name: {X_shifted.columns}")


    columns = ['clear_sky_energy_1h:J', 'diffuse_rad_1h:J', 'direct_rad_1h:J',
               'fresh_snow_12h:cm', 'fresh_snow_1h:cm', 'fresh_snow_24h:cm',
               'fresh_snow_3h:cm', 'fresh_snow_6h:cm']

    # Filter rows where index.minute == 0
    X_shifted = X[X.index.minute == 0][columns].copy()

    # Create a set for constant-time lookup
    index_set = set(X.index)
```

```
    # Vectorized time shifting
    one_hour = pd.Timedelta('1 hour')
    shifted_indices = X_shifted.index + one_hour
    X_shifted.loc[shifted_indices.isin(index_set)] = X.
 ↪loc[shifted_indices[shifted_indices.isin(index_set)]][columns]

    # set last row to same as second last row
    X_shifted.iloc[-1] = X_shifted.iloc[-2]

    # Count
    count1 = len(shifted_indices[shifted_indices.isin(index_set)])
    count2 = len(X_shifted) - count1

    print("COUNT1", count1)
    print("COUNT2", count2)

    # Rename columns
    X_old_unshifted = X_shifted.copy()
    X_old_unshifted.columns = [f"{col}_not_shifted" for col in X_old_unshifted.
 ↪columns]

    date_calc = None
    # If 'date_calc' is present, handle it
    if 'date_calc' in X.columns:
        date_calc = X[X.index.minute == 0]['date_calc']




    # resample to hourly
    print("index: ", X.index[0])
    X = X.resample('H').mean()
    print("index AFTER: ", X.index[0])

    X[columns] = X_shifted[columns]
    #X[X_old_unshifted.columns] = X_old_unshifted

    if date_calc is not None:
        X['date_calc'] = date_calc

    return X




def fix_X(X, name):
```

```python
    # Convert 'date_forecast' to datetime format and replace original column␣
 ↪with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)


    X = feature_engineering(X)

    return X




def handle_features(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_X(X_train_observed, "X_train_observed")
    X_train_estimated = fix_X(X_train_estimated, "X_train_estimated")
    X_test = fix_X(X_test, "X_test")


    if weight_evaluation:
        # add sample weights, which are 1 for observed and 3 for estimated
        X_train_observed["sample_weight"] = 1
        X_train_estimated["sample_weight"] = sample_weight_estimated
        X_test["sample_weight"] = sample_weight_estimated


    y_train['ds'] = pd.to_datetime(y_train['time'])
    y_train.drop(columns=['time'], inplace=True)
    y_train.sort_values(by='ds', inplace=True)
    y_train.set_index('ds', inplace=True)

    return X_train_observed, X_train_estimated, X_test, y_train





def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,␣
 ↪location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train =␣
 ↪handle_features(X_train_observed, X_train_estimated, X_test, y_train)

    if use_estimated_diff_attr:
        X_train_observed["estimated_diff_hours"] = 0
        X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index -␣
 ↪pd.to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
```

```python
        X_test["estimated_diff_hours"] = (X_test.index - pd.to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600

        X_train_estimated["estimated_diff_hours"] = X_train_estimated["estimated_diff_hours"].astype('int64')
        # the filled once will get dropped later anyways, when we drop y nans
        X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].fillna(-50).astype('int64')

    if use_is_estimated_attr:
        X_train_observed["is_estimated"] = 0
        X_train_estimated["is_estimated"] = 1
        X_test["is_estimated"] = 1


    # drop date_calc
    X_train_estimated.drop(columns=['date_calc'], inplace=True)
    X_test.drop(columns=['date_calc'], inplace=True)


    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
    X_train = pd.concat([X_train_observed, X_train_estimated])


    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)

    X_train = pd.merge(X_train, y_train, how="inner", left_index=True, right_index=True)

    # print number of nans in y
    print(f"Number of nans in y: {X_train['y'].isna().sum()}")

    print(f"Size of estimated after dropping nans: {len(X_train[X_train['is_estimated']==1].dropna(subset=['y']))}")


    X_train["location"] = location
    X_test["location"] = location

    return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []
```

```python
# Loop through locations
for loc in locations:
    print(f"Processing location {loc}...")
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed= pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,␣
 ↪X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)
```

```
Processing location A…
COUNT1 29667
COUNT2 1
index:  2019-06-02 22:00:00
index AFTER:  2019-06-02 22:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 0
Size of estimated after dropping nans: 4418
Processing location B…
COUNT1 29232
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
```

```
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 4
Size of estimated after dropping nans: 3625
Processing location C…
COUNT1 29206
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 6059
Size of estimated after dropping nans: 2954
```

2.1 Feature enginering

2.1.1 Remove anomalies

[3]:
```python
import numpy as np
import pandas as pd


# loop thorugh x train[y], keep track of streaks of same values and replace␣
↪them with nan if they are too long
# also replace nan with 0

import numpy as np

def replace_streaks_with_nan(df, max_streak_length, column="y"):
    for location in df["location"].unique():
        x = df[df["location"] == location][column].copy()

        last_val = None
        streak_length = 1
        streak_indices = []
        allowed = [0]
        found_streaks = {}
```

```
        for idx in x.index:
            value = x[idx]
            # if location == "B":
            #     continue

            if value == last_val and value not in allowed:
                streak_length += 1
                streak_indices.append(idx)
            else:
                streak_length = 1
                last_val = value
                streak_indices.clear()

            if streak_length > max_streak_length:
                found_streaks[value] = streak_length

                for streak_idx in streak_indices:
                    x[idx] = np.nan
                streak_indices.clear()  # clear after setting to NaN to avoid␣
 ↪setting multiple times
        df.loc[df["location"] == location, column] = x

        print(f"Found streaks for location {location}: {found_streaks}")

    return df


# deep copy of X_train into x_copy
X_train = replace_streaks_with_nan(X_train.copy(), 3, "y")
```

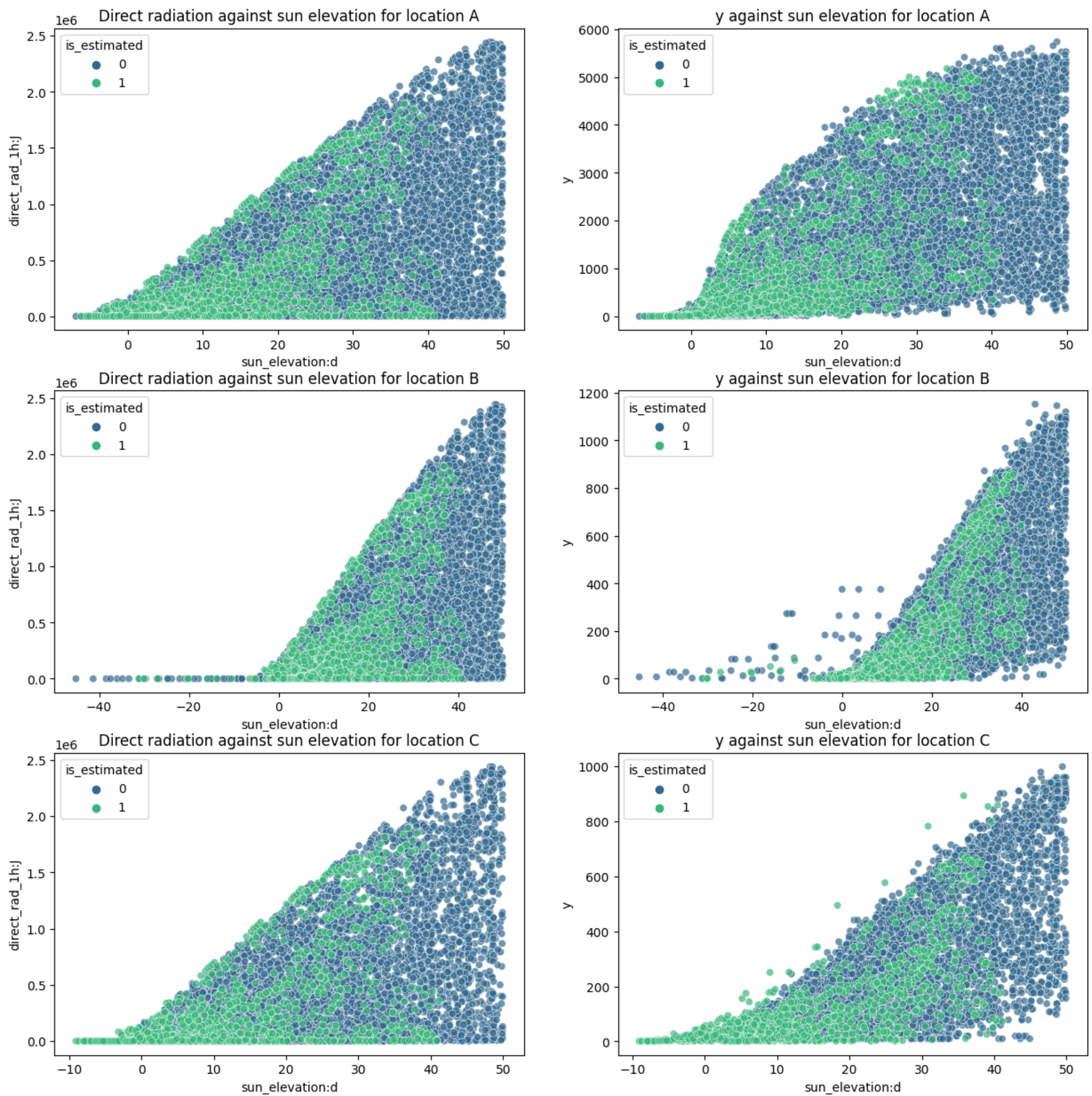
```
Found streaks for location A: {}
Found streaks for location B: {3.45: 28, 6.9: 7, 12.9375: 5, 13.8: 8, 276.0: 78,
18.975: 58, 0.8625: 4, 118.1625: 33, 34.5: 11, 183.7125: 1058, 87.1125: 7,
79.35: 34, 7.7625: 12, 27.6: 448, 273.41249999999997: 72, 264.78749999999997:
55, 169.05: 33, 375.1875: 56, 314.8125: 66, 76.7625: 10, 135.4125: 216, 81.9375:
202, 2.5875: 12, 81.075: 210}
Found streaks for location C: {9.8: 4, 29.400000000000002: 4, 19.6: 4}
```

[4]:
```
# print num rows
temprows = len(X_train)
X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],␣
 ↪inplace=True)
print("Dropped rows: ", temprows - len(X_train))
```

```
Dropped rows:  9293
```

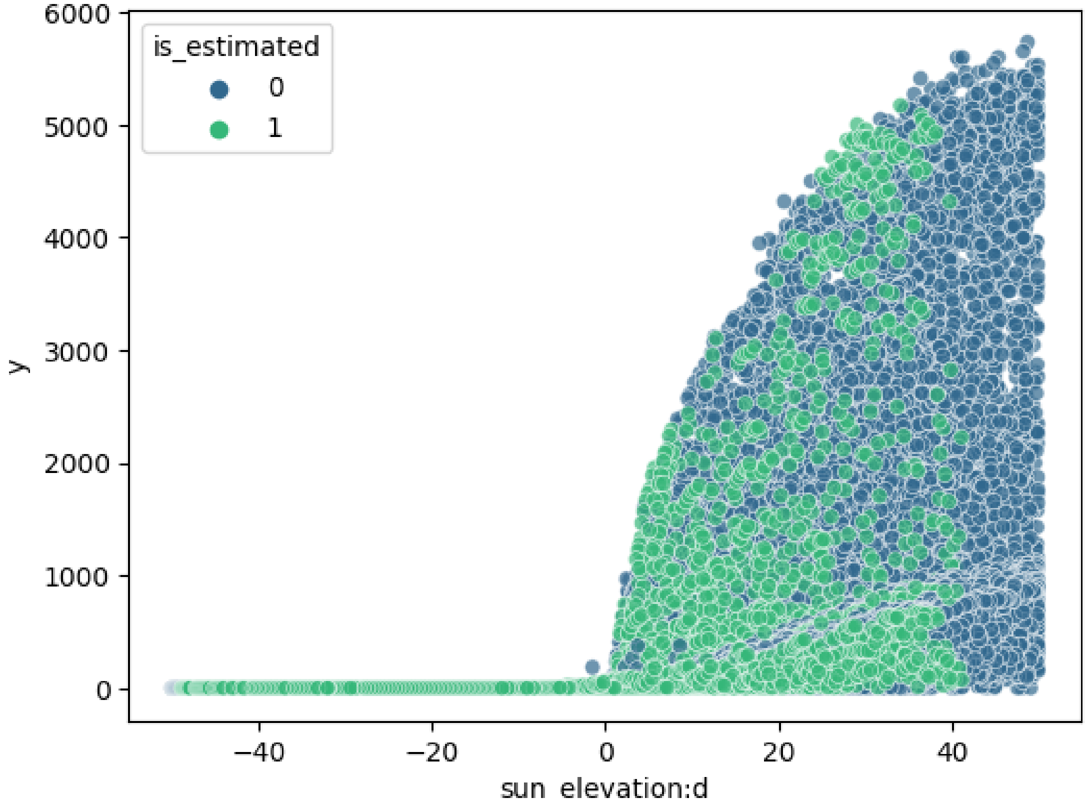
[5]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
# Filter out rows where y == 0
temp = X_train[X_train["y"] != 0]

# Plotting
fig, axes = plt.subplots(len(locations), 2, figsize=(15, 5 * len(locations)))

for idx, location in enumerate(locations):
    sns.scatterplot(ax=axes[idx][0], data=temp[temp["location"] == location],␣
 ↪x="sun_elevation:d", y="direct_rad_1h:J", hue="is_estimated",␣
 ↪palette="viridis", alpha=0.7)
    axes[idx][0].set_title(f"Direct radiation against sun elevation for␣
 ↪location {location}")

    sns.scatterplot(ax=axes[idx][1], data=temp[temp["location"] == location],␣
 ↪x="sun_elevation:d", y="y", hue="is_estimated", palette="viridis", alpha=0.7)
    axes[idx][1].set_title(f"y against sun elevation for location {location}")

# plt.tight_layout()
# plt.show()
```



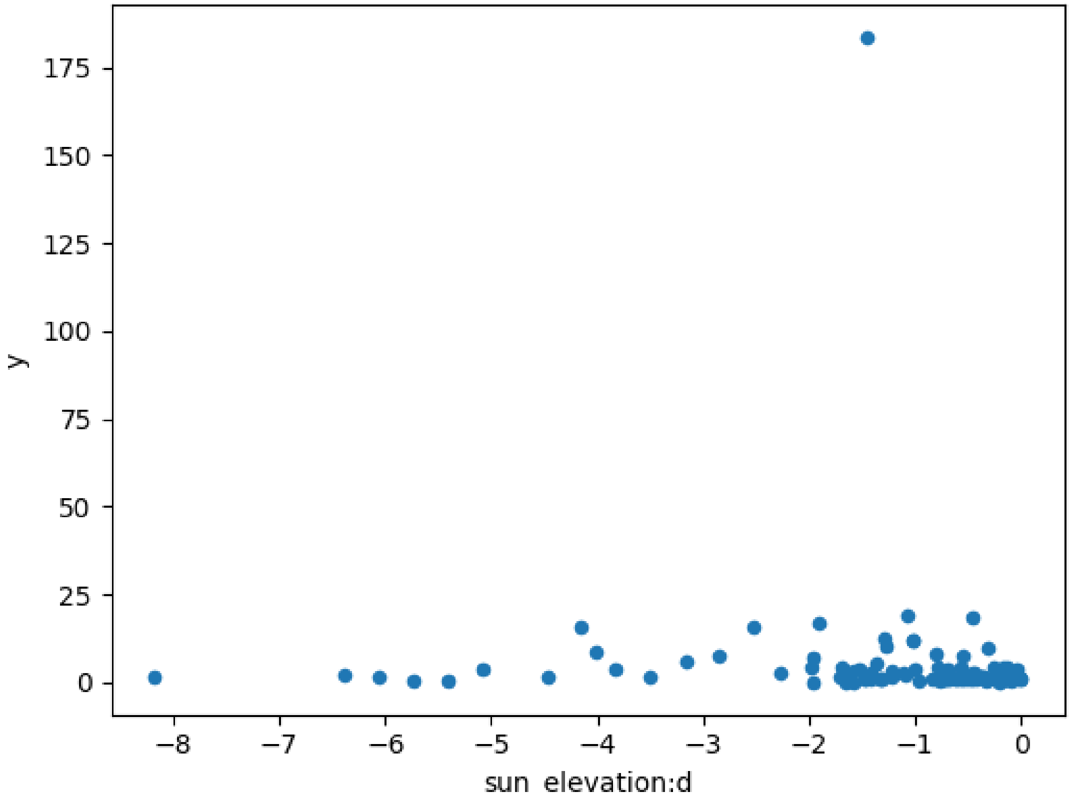
[6]:

```
thresh = 0.1

# Update "y" values to NaN if they don't meet the criteria
mask = (X_train["direct_rad_1h:J"] <= thresh) & (X_train["diffuse_rad_1h:J"] <=␣
 ↪thresh) & (X_train["y"] >= 0.1)
if drop_night_outliers:
    X_train.loc[mask, "y"] = np.nan

# Plot using sns scatterplot
sns.scatterplot(data=X_train, x="sun_elevation:d", y="y", hue="is_estimated",␣
 ↪palette="viridis", alpha=0.7)
plt.show()
```



[7]:
```
# location B count number of rows with y > 0 and sun_elevation:d < 0

condition = (X_train["location"] == "B") & (X_train["y"] > 0) &␣
↪(X_train["sun_elevation:d"] < 0)
bad = X_train[condition]

bad.plot.scatter(x="sun_elevation:d", y="y")
```

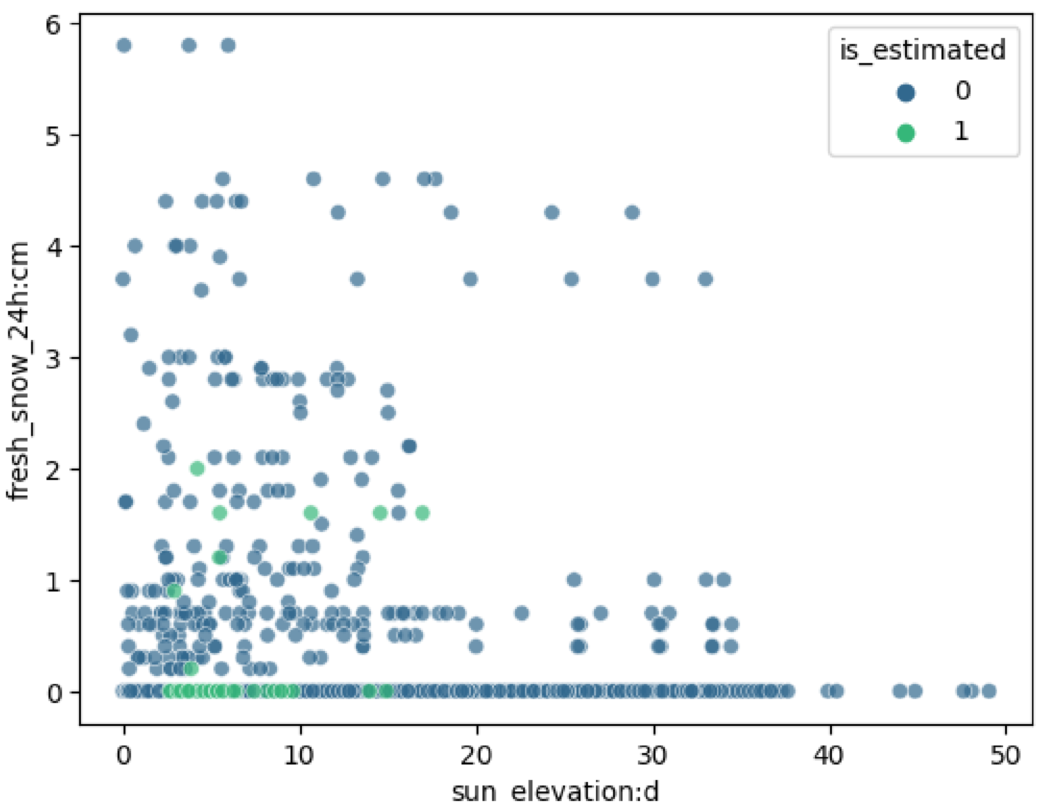
[7]:
```
<AxesSubplot: xlabel='sun_elevation:d', ylabel='y'>
```



[8]:
```
# set y to nan where y is 0, but direct_rad_1h:J or diffuse_rad_1h:J are > 0␣
↪(or some threshold)
threshold_direct = X_train["direct_rad_1h:J"].max() * 0.01
threshold_diffuse = X_train["diffuse_rad_1h:J"].max() * 0.01
print(f"Threshold direct: {threshold_direct}")
print(f"Threshold diffuse: {threshold_diffuse}")

mask = (X_train["y"] == 0) & ((X_train["direct_rad_1h:J"] > threshold_direct) |␣
↪(X_train["diffuse_rad_1h:J"] > threshold_diffuse)) & (X_train["sun_elevation:
↪d"] > 0) & (X_train["fresh_snow_24h:cm"] < 6) & (X_train[['fresh_snow_12h:
↪cm', 'fresh_snow_1h:cm',  'fresh_snow_3h:cm', 'fresh_snow_6h:cm']].
↪sum(axis=1) == 0)
print(len(X_train[mask]))

#print(X_train[mask][[x for x in X_train.columns if "snow" in x]])

# show plot where mask is true
#sns.scatterplot(data=X_train[mask], x="sun_elevation:d", y="y",␣
↪hue="is_estimated", palette="viridis", alpha=0.7)
```

```python
sns.scatterplot(data=X_train[mask], x="sun_elevation:d", y="fresh_snow_24h:cm",␣
    ↪hue="is_estimated", palette="viridis", alpha=0.7)
plt.show()

#sns.scatterplot(data=X_train[mask], x="fresh_snow_24h:cm",␣
    ↪y="total_cloud_cover:p", hue="is_estimated", palette="viridis", alpha=0.7)



# set y to nan where mask
if drop_null_outliers:
    X_train.loc[mask, "y"] = np.nan

# show how many rows for each location, and for estimated and not estimated
X_train[mask].groupby(["location", "is_estimated"]).count()["direct_rad_1h:J"]
```

```
Threshold direct: 24458.97
Threshold diffuse: 11822.505000000001
2599
```



[8]:
```
location  is_estimated
A         0                 87
          1                 10
B         0               1250
          1                 32
C         0               1174
          1                 46
Name: direct_rad_1h:J, dtype: int64
```

[9]:
```python
# print num rows
temprows = len(X_train)
X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],
 ↪inplace=True)
print("Dropped rows: ", temprows - len(X_train))
```

```
Dropped rows:  1876
```

2.1.2 Other stuff

[10]:
```python
import numpy as np
import pandas as pd

for attr in use_dt_attrs:
    X_train[attr] = getattr(X_train.index, attr)
    X_test[attr] = getattr(X_test.index, attr)

#print(X_train.head())


# If the "sample_weight" column is present and weight_evaluation is True,
 ↪multiply sample_weight with sample_weight_may_july if the ds is between
 ↪05-01 00:00:00 and 07-03 23:00:00, else add sample_weight as a column to
 ↪X_train
if weight_evaluation:
    if "sample_weight" not in X_train.columns:
        X_train["sample_weight"] = 1

    X_train.loc[((X_train.index.month >= 5) & (X_train.index.month <= 6)) |
 ↪((X_train.index.month == 7) & (X_train.index.day <= 3)), "sample_weight"] *=
 ↪sample_weight_may_july


print(X_train.iloc[200])
print(X_train[((X_train.index.month >= 5) & (X_train.index.month <= 6)) |
 ↪((X_train.index.month == 7) & (X_train.index.day <= 3))].head(1))
```

```
if use_groups:
    # fix groups for cross validation
    locations = X_train['location'].unique()  # Assuming 'location' is the name␣
 ↪of the column representing locations

    grouped_dfs = []  # To store data frames split by location

    # Loop through each unique location
    for loc in locations:
        loc_df = X_train[X_train['location'] == loc]

        # Sort the DataFrame for this location by the time column
        loc_df = loc_df.sort_index()

        # Calculate the size of each group for this location
        group_size = len(loc_df) // n_groups

        # Create a new 'group' column for this location
        loc_df['group'] = np.repeat(range(n_groups),␣
 ↪repeats=[group_size]*(n_groups-1) + [len(loc_df) - group_size*(n_groups-1)])

        # Append to list of grouped DataFrames
        grouped_dfs.append(loc_df)

    # Concatenate all the grouped DataFrames back together
    X_train = pd.concat(grouped_dfs)
    X_train.sort_index(inplace=True)
    print(X_train["group"].head())






X_train.drop(columns=to_drop, inplace=True)
X_test.drop(columns=to_drop, inplace=True)

X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)
```

```
absolute_humidity_2m:gm3              7.625
air_density_2m:kgm3                  1.2215
ceiling_height_agl:m            3644.050049
clear_sky_energy_1h:J            2896336.75
clear_sky_rad:W                  753.849976
cloud_base_agl:m                3644.050049
dew_or_rime:idx                         0.0
```

```
dew_point_2m:K                     280.475006
diffuse_rad:W                      127.475006
diffuse_rad_1h:J                   526032.625
direct_rad:W                            488.0
direct_rad_1h:J                   1718048.625
effective_cloud_cover:p             18.200001
elevation:m                               6.0
fresh_snow_12h:cm                         0.0
fresh_snow_1h:cm                          0.0
fresh_snow_24h:cm                         0.0
fresh_snow_3h:cm                          0.0
fresh_snow_6h:cm                          0.0
is_day:idx                                1.0
is_in_shadow:idx                          0.0
msl_pressure:hPa                  1026.775024
precip_5min:mm                            0.0
precip_type_5min:idx                      0.0
pressure_100m:hPa                 1013.599976
pressure_50m:hPa                  1019.599976
prob_rime:p                               0.0
rain_water:kgm2                           0.0
relative_humidity_1000hPa:p         53.825001
sfc_pressure:hPa                  1025.699951
snow_density:kgm3                         NaN
snow_depth:cm                             0.0
snow_drift:idx                            0.0
snow_melt_10min:mm                        0.0
snow_water:kgm2                           0.0
sun_azimuth:d                      222.089005
sun_elevation:d                     44.503498
super_cooled_liquid_water:kgm2            0.0
t_1000hPa:K                        286.700012
total_cloud_cover:p                 18.200001
visibility:m                         52329.25
wind_speed_10m:ms                         2.6
wind_speed_u_10m:ms                      -1.9
wind_speed_v_10m:ms                     -1.75
wind_speed_w_1000hPa:ms                   0.0
is_estimated                                0
y                                     4367.44
location                                    A
Name: 2019-06-11 13:00:00, dtype: object
                     absolute_humidity_2m:gm3  air_density_2m:kgm3  \
ds
2019-06-02 23:00:00                       7.7               1.2235

                     ceiling_height_agl:m  clear_sky_energy_1h:J  \
ds
```

```
2019-06-02 23:00:00           1689.824951                    0.0

                     clear_sky_rad:W  cloud_base_agl:m  dew_or_rime:idx  \
ds
2019-06-02 23:00:00              0.0       1689.824951              0.0

                     dew_point_2m:K  diffuse_rad:W  diffuse_rad_1h:J  …  \
ds                                                                    …
2019-06-02 23:00:00      280.299988            0.0               0.0  …

                     t_1000hPa:K  total_cloud_cover:p  visibility:m  \
ds
2019-06-02 23:00:00   286.899994                100.0  33770.648438

                     wind_speed_10m:ms  wind_speed_u_10m:ms  \
ds
2019-06-02 23:00:00               3.35                -3.35

                     wind_speed_v_10m:ms  wind_speed_w_1000hPa:ms  \
ds
2019-06-02 23:00:00                0.275                      0.0

                     is_estimated    y  location
ds
2019-06-02 23:00:00             0  0.0         A

[1 rows x 48 columns]
```

[11]:

```python
# Create a plot of X_train showing its "y" and color it based on the value of␣
↪the sample_weight column.
if "sample_weight" in X_train.columns:
    import matplotlib.pyplot as plt
    import seaborn as sns
    sns.scatterplot(data=X_train, x=X_train.index, y="y", hue="sample_weight",␣
↪palette="deep", size=3)
    plt.show()
```

[12]:

```python
def normalize_sample_weights_per_location(df):
    for loc in locations:
        loc_df = df[df["location"] == loc]
        loc_df["sample_weight"] = loc_df["sample_weight"] /␣
↪loc_df["sample_weight"].sum() * loc_df.shape[0]
        df[df["location"] == loc] = loc_df
    return df

import pandas as pd
```

```python
def split_and_shuffle_data(input_data, num_bins, frac1):
    """
    Splits the input_data into num_bins and shuffles them, then divides the bins into two datasets based on the given fraction for the first set.

    Args:
        input_data (pd.DataFrame): The data to be split and shuffled.
        num_bins (int): The number of bins to split the data into.
        frac1 (float): The fraction of each bin to go into the first output dataset.

    Returns:
        pd.DataFrame, pd.DataFrame: The two output datasets.
    """
    # Validate the input fraction
    if frac1 < 0 or frac1 > 1:
        raise ValueError("frac1 must be between 0 and 1.")

    if frac1==1:
        return input_data, pd.DataFrame()

    # Calculate the fraction for the second output set
    frac2 = 1 - frac1

    # Calculate bin size
    bin_size = len(input_data) // num_bins

    # Initialize empty DataFrames for output
    output_data1 = pd.DataFrame()
    output_data2 = pd.DataFrame()

    for i in range(num_bins):
        # Shuffle the data in the current bin
        np.random.seed(i*10)
        current_bin = input_data.iloc[i * bin_size: (i + 1) * bin_size].sample(frac=1)

        # Calculate the sizes for each output set
        size1 = int(len(current_bin) * frac1)

        # Split and append to output DataFrames
        output_data1 = pd.concat([output_data1, current_bin.iloc[:size1]])
        output_data2 = pd.concat([output_data2, current_bin.iloc[size1:]])

    # Shuffle and split the remaining data
    remaining_data = input_data.iloc[num_bins * bin_size:].sample(frac=1)
```

```
    remaining_size1 = int(len(remaining_data) * frac1)

    output_data1 = pd.concat([output_data1, remaining_data.iloc[:
↪remaining_size1]])
    output_data2 = pd.concat([output_data2, remaining_data.iloc[remaining_size1:
↪]])

    return output_data1, output_data2
```

[13]:
```
from autogluon.tabular import TabularDataset, TabularPredictor
data = TabularDataset('X_train_raw.csv')
# set group column of train_data be increasing from 0 to 7 based on time, the␣
↪first 1/8 of the data is group 0, the second 1/8 of the data is group 1, etc.
data['ds'] = pd.to_datetime(data['ds'])
data = data.sort_values(by='ds')

# # print size of the group for each location
# for loc in locations:
#     print(f"Location {loc}:")
#     print(train_data[train_data["location"] == loc].groupby('group').size())


# get end date of train data and subtract 3 months
#split_time = pd.to_datetime(train_data["ds"]).max() - pd.
↪Timedelta(hours=tune_and_test_length)
# 2022-10-28 22:00:00
split_time = pd.to_datetime("2022-10-28 22:00:00")
train_set = TabularDataset(data[data["ds"] < split_time])
estimated_set = TabularDataset(data[data["ds"] >= split_time]) # only estimated

test_set = pd.DataFrame()
tune_set = pd.DataFrame()
new_train_set = pd.DataFrame()

if not use_tune_data:
    raise Exception("Not implemented")

for location in locations:
    loc_data = data[data["location"] == location]
    num_train_rows = len(loc_data)

    tune_rows = 1500.0 # 2500.0
    if use_test_data:
        tune_rows = 1880.0#max(3000.0,␣
↪len(estimated_set[estimated_set["location"] == location]))
```

```
    holdout_frac = max(0.01, min(0.1, tune_rows / num_train_rows)) *␣
↪num_train_rows / len(estimated_set[estimated_set["location"] == location])

    print(f"Size of estimated for location {location}:␣
↪{len(estimated_set[estimated_set['location'] == location])}. Holdout frac␣
↪should be % of estimated: {holdout_frac}")

    # shuffle and split data
    loc_tune_set, loc_new_train_set =␣
↪split_and_shuffle_data(estimated_set[estimated_set['location'] == location],␣
↪40, holdout_frac)
    print(f"Length of location tune set : {len(loc_tune_set)}")
    new_train_set = pd.concat([new_train_set, loc_new_train_set])

    if use_test_data:
        loc_test_set, loc_tune_set = split_and_shuffle_data(loc_tune_set, 40, 0.
↪2)
        test_set = pd.concat([test_set, loc_test_set])

    tune_set = pd.concat([tune_set, loc_tune_set])




print("Length of train set before adding test set", len(train_set))
# add rest to train_set
train_set = pd.concat([train_set, new_train_set])
print("Length of train set after adding test set", len(train_set))




if use_groups:
    test_set = test_set.drop(columns=['group'])


tuning_data = tune_set

# number of rows in tuning data for each location
print("Shapes of tuning data", tuning_data.groupby('location').size())



if use_test_data:
    test_data = test_set
    print("Shape of test", test_data.shape[0])
```

```
train_data = train_set

# ensure sample weights for your training (or tuning) data sum to the number of␣
↪rows in the training (or tuning) data.
if weight_evaluation:
    # ensure sample weights for data sum to the number of rows in the tuning /
↪train data.
    tuning_data = normalize_sample_weights_per_location(tuning_data)
    train_data = normalize_sample_weights_per_location(train_data)
    if use_test_data:
        test_data = normalize_sample_weights_per_location(test_data)


train_data = TabularDataset(train_data)
tuning_data = TabularDataset(tuning_data)

if use_test_data:
    test_data = TabularDataset(test_data)
```

```
Size of estimated for location A: 4214. Holdout frac should be % of estimated:
0.4461319411485524
Length of location tune set : 1846
Size of estimated for location B: 3533. Holdout frac should be % of estimated:
0.5321256722332296
Length of location tune set : 1846
Size of estimated for location C: 2923. Holdout frac should be % of estimated:
0.6431748203900103
Length of location tune set : 1841
Length of train set before adding test set 77247
Length of train set after adding test set 82384
Shapes of tuning data location
A    1485
B    1485
C    1481
dtype: int64
Shape of test 1082
```

3 Quick EDA

[14]:
```
if run_analysis:
    import autogluon.eda.auto as auto
    auto.dataset_overview(train_data=train_data, test_data=test_data,␣
↪label="y", sample=None)
```

[15]:
```
if run_analysis:
    auto.target_analysis(train_data=train_data, label="y", sample=None)
```

4 Modeling

[16]:
```
import os


# Get the last submission number
last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for␣
↪filename in os.listdir('submissions') if "submission" in filename]))
print("Last submission number:", last_submission_number)
print("Now creating submission number:", last_submission_number + 1)

# Create the new filename
new_filename = f'submission_{last_submission_number + 1}'

hello = os.environ.get('HELLO')
if hello is not None:
    new_filename += f'_{hello}'

print("New filename:", new_filename)
```

```
Last submission number: 119
Now creating submission number: 120
New filename: submission_120
```

[17]:
```
predictors = [None, None, None]
```

[18]:
```
def fit_predictor_for_location(loc):
    print(f"Training model for location {loc}...")
    # sum of sample weights for this location, and number of rows, for both␣
↪train and tune data and test data
    if weight_evaluation:
        print("Train data sample weight sum:",␣
↪train_data[train_data["location"] == loc]["sample_weight"].sum())
        print("Train data number of rows:", train_data[train_data["location"]␣
↪== loc].shape[0])
        if use_tune_data:
            print("Tune data sample weight sum:",␣
↪tuning_data[tuning_data["location"] == loc]["sample_weight"].sum())
            print("Tune data number of rows:",␣
↪tuning_data[tuning_data["location"] == loc].shape[0])
        if use_test_data:
            print("Test data sample weight sum:",␣
↪test_data[test_data["location"] == loc]["sample_weight"].sum())
            print("Test data number of rows:", test_data[test_data["location"]␣
↪== loc].shape[0])
    predictor = TabularPredictor(
        label=label,
```

```
        eval_metric=metric,
        path=f"AutogluonModels/{new_filename}_{loc}",
        # sample_weight=sample_weight,
        # weight_evaluation=weight_evaluation,
        # groups="group" if use_groups else None,
    ).fit(
        train_data=train_data[train_data["location"] == loc].
↪drop(columns=["ds"]),
        time_limit=time_limit,
        presets=presets,
        num_stack_levels=num_stack_levels,
        num_bag_folds=num_bag_folds if not use_groups else 2,# just put␣
↪somethin, will be overwritten anyways
        num_bag_sets=num_bag_sets,
        tuning_data=tuning_data[tuning_data["location"] == loc].
↪reset_index(drop=True).drop(columns=["ds"]) if use_tune_data else None,
        use_bag_holdout=use_bag_holdout,
        # holdout_frac=holdout_frac,
    )

    # evaluate on test data
    if use_test_data:
        # drop sample_weight column
        t = test_data[test_data["location"] == loc]#.
↪drop(columns=["sample_weight"])
        perf = predictor.evaluate(t)
        print("Evaluation on test data:")
        print(perf[predictor.eval_metric.name])

    return predictor

loc = "A"
predictors[0] = fit_predictor_for_location(loc)
```

```
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=0, num_bag_folds=8,
num_bag_sets=20
Beginning AutoGluon training … Time limit = 600s
AutoGluon will save models to "AutogluonModels/submission_120_A/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   171.00 GB / 315.93 GB (54.1%)
Train Data Rows:    30934
Train Data Columns: 44
```

```
Tuning Data Rows:    1485
Tuning Data Columns: 44
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (5733.42, 0.0, 673.30495,
1195.09297)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    132218.01 MB
        Train Data (Original)  Memory Usage: 13.03 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 2 features to boolean dtype as they
only contain 2 unique values.

Training model for location A…

        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 3): ['elevation:m', 'snow_drift:idx',
'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 40 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 39 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', ['bool']) :  2 | ['snow_density:kgm3', 'is_estimated']
```

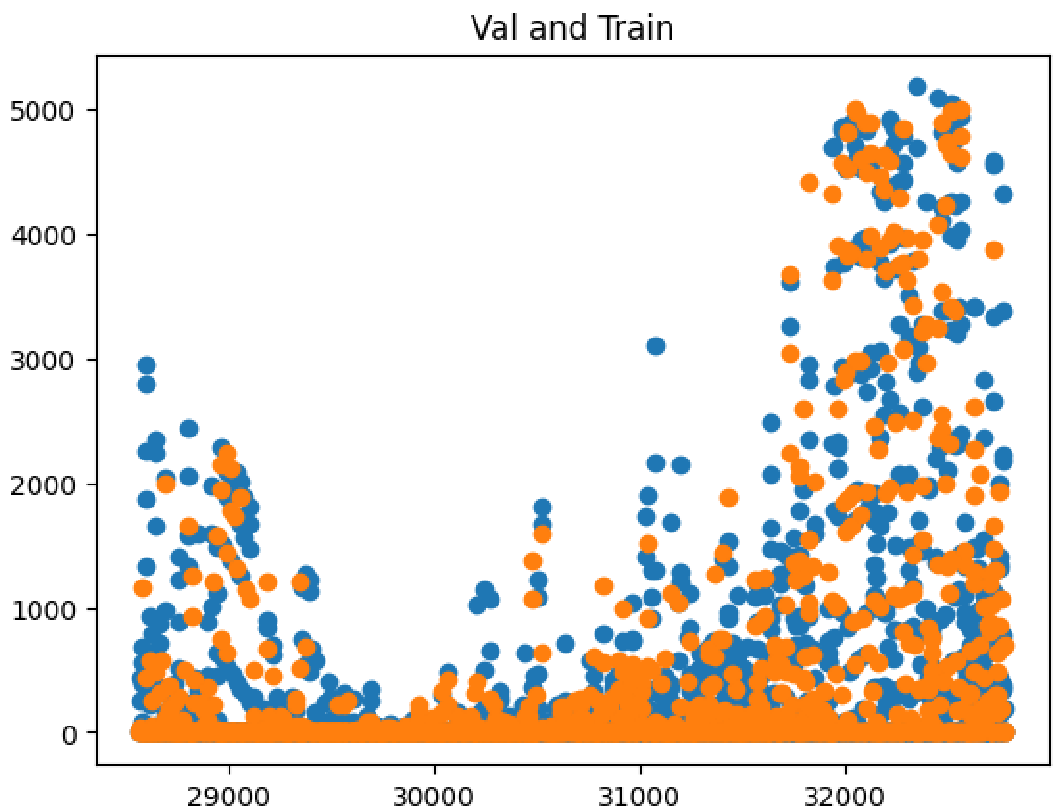
```
        0.2s = Fit runtime
        41 features in original data used to generate 41 features in processed
data.
        Train Data (Processed) Memory Usage: 10.18 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.18s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 599.82s of the
599.82s of remaining time.
        -190.7667        = Validation score   (-mean_absolute_error)
        0.04s    = Training   runtime
        0.39s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 599.3s of the
599.3s of remaining time.
        -192.1817        = Validation score   (-mean_absolute_error)
        0.04s    = Training   runtime
        0.39s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 598.8s of the
598.8s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
```

```
        -83.0011         = Validation score   (-mean_absolute_error)
        32.02s   = Training   runtime
        14.18s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 557.51s of the
557.51s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -92.2921         = Validation score   (-mean_absolute_error)
        25.22s   = Training   runtime
        5.32s    = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 528.76s of
the 528.76s of remaining time.
        -103.3885        = Validation score   (-mean_absolute_error)
        8.32s    = Training   runtime
        1.12s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 518.17s of the
518.16s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -97.761  = Validation score   (-mean_absolute_error)
        208.44s  = Training   runtime
        0.1s     = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 308.57s of the
308.56s of remaining time.
        -104.3559        = Validation score   (-mean_absolute_error)
        1.75s    = Training   runtime
        1.1s     = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 304.59s of
the 304.58s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -102.97  = Validation score   (-mean_absolute_error)
        37.88s   = Training   runtime
        0.48s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 264.08s of the
264.08s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -97.2924         = Validation score   (-mean_absolute_error)
        59.12s   = Training   runtime
        4.27s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 200.4s of the
200.4s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -86.4137         = Validation score   (-mean_absolute_error)
        128.94s  = Training   runtime
        0.41s    = Validation runtime
```

```
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 70.05s of the
70.05s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -90.8486         = Validation score   (-mean_absolute_error)
        59.44s    = Training   runtime
        17.27s    = Validation runtime
Completed 1/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
2.88s of remaining time.
        -81.2113         = Validation score   (-mean_absolute_error)
        0.41s     = Training   runtime
        0.0s      = Validation runtime
AutoGluon training complete, total runtime = 597.56s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_120_A/")
Evaluation: mean_absolute_error on test data: -77.99246722137123
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -77.99246722137123,
    "root_mean_squared_error": -226.17935500275425,
    "mean_squared_error": -51157.10062946194,
    "r2": 0.9296313105079771,
    "pearsonr": 0.9646281962110672,
    "median_absolute_error": -3.7184290885925293
}

Evaluation on test data:
-77.99246722137123
```

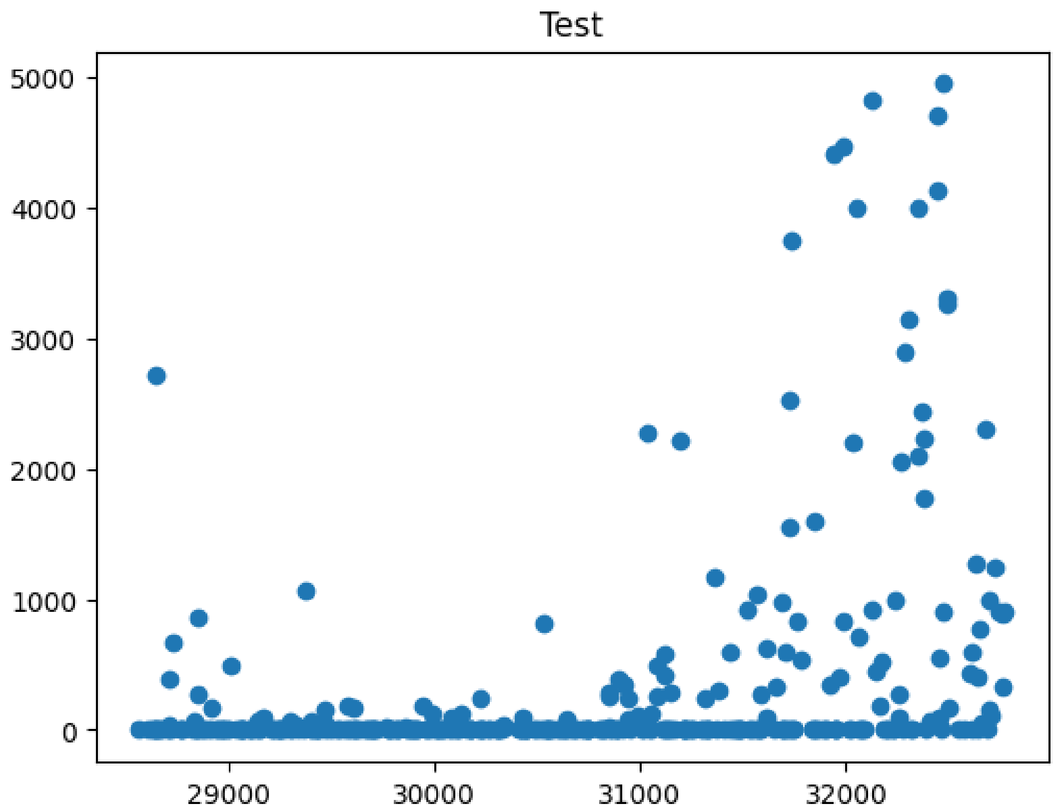
[19]:
```python
import matplotlib.pyplot as plt
leaderboards = [None, None, None]
def leaderboard_for_location(i, loc):
    if use_tune_data:
        plt.scatter(train_data[(train_data["location"] == loc) &␣
 ↪(train_data["is_estimated"]==True)]["y"].index,␣
 ↪train_data[(train_data["location"] == loc) &␣
 ↪(train_data["is_estimated"]==True)]["y"])
        plt.scatter(tuning_data[tuning_data["location"] == loc]["y"].index,␣
 ↪tuning_data[tuning_data["location"] == loc]["y"])
        plt.title("Val and Train")
        plt.show()

        if use_test_data:
```

```
        lb = predictors[i].leaderboard(test_data[test_data["location"] ==␣
↪loc])
        lb["location"] = loc
        plt.scatter(test_data[test_data["location"] == loc]["y"].index,␣
↪test_data[test_data["location"] == loc]["y"])
        plt.title("Test")

        return lb

    return pd.DataFrame()

leaderboards[0] = leaderboard_for_location(0, loc)
```




```
                      model  score_test   score_val  pred_time_test
pred_time_val    fit_time  pred_time_test_marginal  pred_time_val_marginal
fit_time_marginal  stack_level  can_infer  fit_order
0       WeightedEnsemble_L2  -77.992467  -81.211309        1.084604
14.589731  161.374790                 0.002805                0.000588
0.414011            2       True         12
1         LightGBMXT_BAG_L1  -81.295824  -83.001111        0.885789
14.181989   32.023941                 0.885789               14.181989
```

```
32.023941            1       True          3
2     LightGBMLarge_BAG_L1  -82.805017  -90.848551        1.917668
17.274774   59.435563                1.917668               17.274774
59.435563            1       True          11
3    NeuralNetTorch_BAG_L1  -85.166614  -86.413694        0.196011
0.407153  128.936838                0.196011               0.407153
128.936838            1       True          10
4          LightGBM_BAG_L1  -86.706664  -92.292094        0.528157
5.320704   25.218896                0.528157               5.320704
25.218896            1       True          4
5          CatBoost_BAG_L1  -91.111087  -97.761035        0.064080
0.095527  208.440814                0.064080               0.095527
208.440814            1       True          6
6           XGBoost_BAG_L1  -91.761543  -97.292393        0.717162
4.265924   59.117398                0.717162               4.265924
59.117398            1       True          9
7   RandomForestMSE_BAG_L1  -94.975608 -103.388512        0.538202
1.116199    8.324562                0.538202               1.116199
8.324562            1       True          5
8     ExtraTreesMSE_BAG_L1  -95.069663 -104.355896        0.530033
1.097124    1.747186                0.530033               1.097124
1.747186            1       True          7
9   NeuralNetFastAI_BAG_L1 -101.305052 -102.969968        0.188707
0.484712   37.876854                0.188707               0.484712
37.876854            1       True          8
10   KNeighborsUnif_BAG_L1 -177.356321 -190.766696        0.166278
0.392006    0.035234                0.166278               0.392006
0.035234            1       True          1
11   KNeighborsDist_BAG_L1 -182.556761 -192.181679        0.012831
0.389233    0.035717                0.012831               0.389233
0.035717            1       True          2
```



[20]:
```
loc = "B"
predictors[1] = fit_predictor_for_location(loc)
leaderboards[1] = leaderboard_for_location(1, loc)
```

```
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=0, num_bag_folds=8,
num_bag_sets=20
Beginning AutoGluon training … Time limit = 600s
AutoGluon will save models to "AutogluonModels/submission_120_B/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   169.06 GB / 315.93 GB (53.5%)
Train Data Rows:    27377
Train Data Columns: 44
Tuning Data Rows:     1485
Tuning Data Columns: 44
Label Column: y
Preprocessing data …
```

```
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (1152.3, -0.0, 97.72983, 206.09638)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])

Training model for location B…

Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    130156.34 MB
        Train Data (Original)  Memory Usage: 11.6 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 2 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 41 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 40 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', ['bool']) :  2 | ['snow_density:kgm3', 'is_estimated']
        0.2s = Fit runtime
        42 features in original data used to generate 42 features in processed
data.
        Train Data (Processed) Memory Usage: 9.29 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.2s …
AutoGluon will gauge predictive performance using evaluation metric:
```

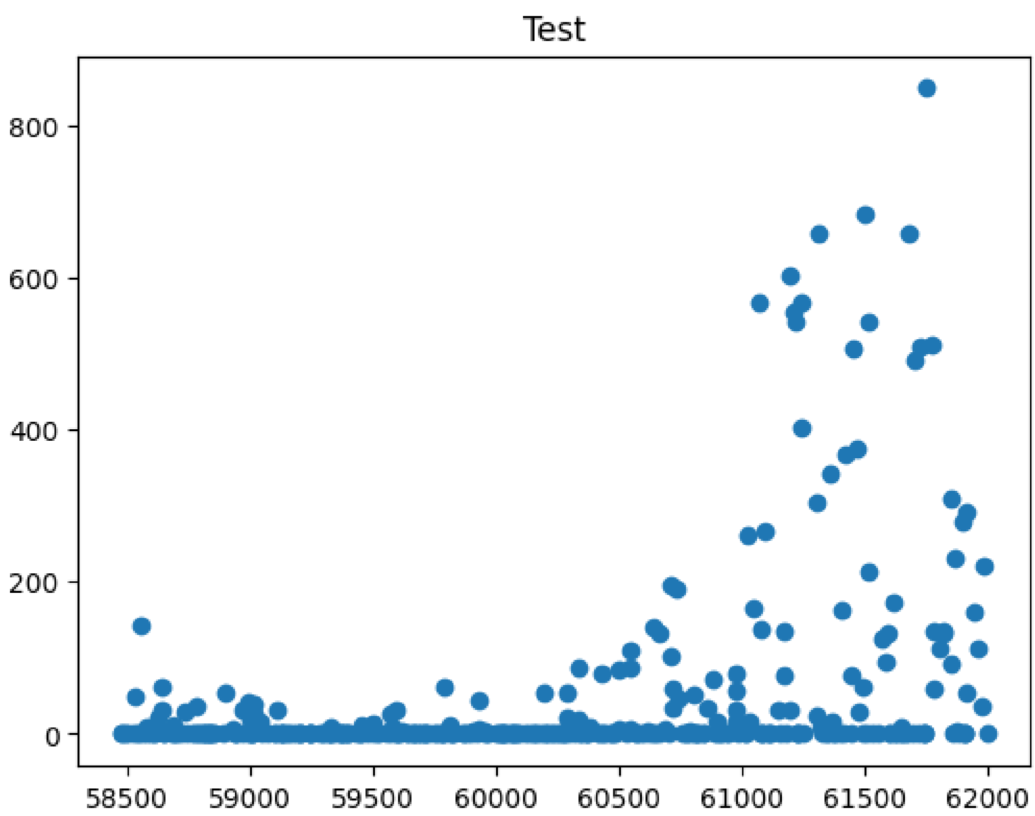
```
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 599.79s of the
599.79s of remaining time.
        -30.1965         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.35s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 599.35s of the
599.35s of remaining time.
        -30.1657         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.36s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 598.88s of the
598.88s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.5747         = Validation score   (-mean_absolute_error)
        29.79s   = Training   runtime
        17.55s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 563.01s of the
563.01s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
```

```
ParallelLocalFoldFittingStrategy
        -14.9758         = Validation score   (-mean_absolute_error)
        33.85s   = Training   runtime
        11.6s    = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 525.1s of the
525.1s of remaining time.
        -16.7143         = Validation score   (-mean_absolute_error)
        6.88s    = Training   runtime
        0.92s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 516.5s of the 516.5s
of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.6218         = Validation score   (-mean_absolute_error)
        200.05s  = Training   runtime
        0.09s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 315.2s of the
315.19s of remaining time.
        -15.7274         = Validation score   (-mean_absolute_error)
        1.43s    = Training   runtime
        0.93s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 312.0s of the
312.0s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.7105         = Validation score   (-mean_absolute_error)
        34.78s   = Training   runtime
        0.48s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 275.65s of the
275.64s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.9131         = Validation score   (-mean_absolute_error)
        78.65s   = Training   runtime
        6.36s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 193.09s of the
193.09s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.2378         = Validation score   (-mean_absolute_error)
        123.54s  = Training   runtime
        0.36s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 68.14s of the
68.14s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.4345         = Validation score   (-mean_absolute_error)
        57.62s   = Training   runtime
```

```
        15.52s   = Validation runtime
Completed 1/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
3.08s of remaining time.
        -12.5371         = Validation score   (-mean_absolute_error)
        0.41s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 597.36s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_120_B/")
Evaluation: mean_absolute_error on test data: -10.869228935804836
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -10.869228935804836,
    "root_mean_squared_error": -32.82364319065276,
    "mean_squared_error": -1077.3915523072853,
    "r2": 0.9337436842186726,
    "pearsonr": 0.9664568667655078,
    "median_absolute_error": -0.3785351812839508
}

Evaluation on test data:
-10.869228935804836
```


```
                    model  score_test  score_val  pred_time_test  pred_time_val
fit_time  pred_time_test_marginal  pred_time_val_marginal  fit_time_marginal
stack_level  can_infer  fit_order
0      WeightedEnsemble_L2  -10.869229 -12.537080        1.652721      19.316098
189.955817                 0.003775                0.000588           0.412009
2       True          12
1    NeuralNetTorch_BAG_L1  -10.927985 -13.237759        0.199645       0.363826
123.543744                 0.199645                0.363826         123.543744
1       True          10
2   NeuralNetFastAI_BAG_L1  -12.586629 -14.710480        0.170884       0.478851
34.784970                 0.170884                0.478851          34.784970
1       True           8
3     LightGBMLarge_BAG_L1  -12.785663 -14.434526        1.724891      15.520442
57.620470                 1.724891               15.520442          57.620470
1       True          11
4        LightGBMXT_BAG_L1  -13.067421 -13.574673        0.903644      17.545081
29.785400                 0.903644               17.545081          29.785400
1       True           3
5           XGBoost_BAG_L1  -13.339945 -14.913071        1.128497       6.363875
78.652484                 1.128497                6.363875          78.652484
1       True           9
```

```
6           CatBoost_BAG_L1  -13.582926 -14.621794         0.065933        0.089616
200.054595                 0.065933                 0.089616         200.054595
1        True           6
7           LightGBM_BAG_L1  -13.821217 -14.975771         0.883035       11.599576
33.846230                 0.883035                11.599576          33.846230
1        True           4
8      ExtraTreesMSE_BAG_L1  -15.113014 -15.727403         0.374773        0.927753
1.429695                 0.374773                 0.927753           1.429695
1        True           7
9    RandomForestMSE_BAG_L1  -15.886907 -16.714303         0.379924        0.921436
6.884286                 0.379924                 0.921436           6.884286
1        True           5
10    KNeighborsUnif_BAG_L1  -28.672377 -30.196526         0.012214        0.345946
0.028718                 0.012214                 0.345946           0.028718
1        True           1
11    KNeighborsDist_BAG_L1  -28.890616 -30.165744         0.016629        0.360982
0.031462                 0.016629                 0.360982           0.031462
1        True           2
```



[21]:
```
loc = "C"
predictors[2] = fit_predictor_for_location(loc)
```

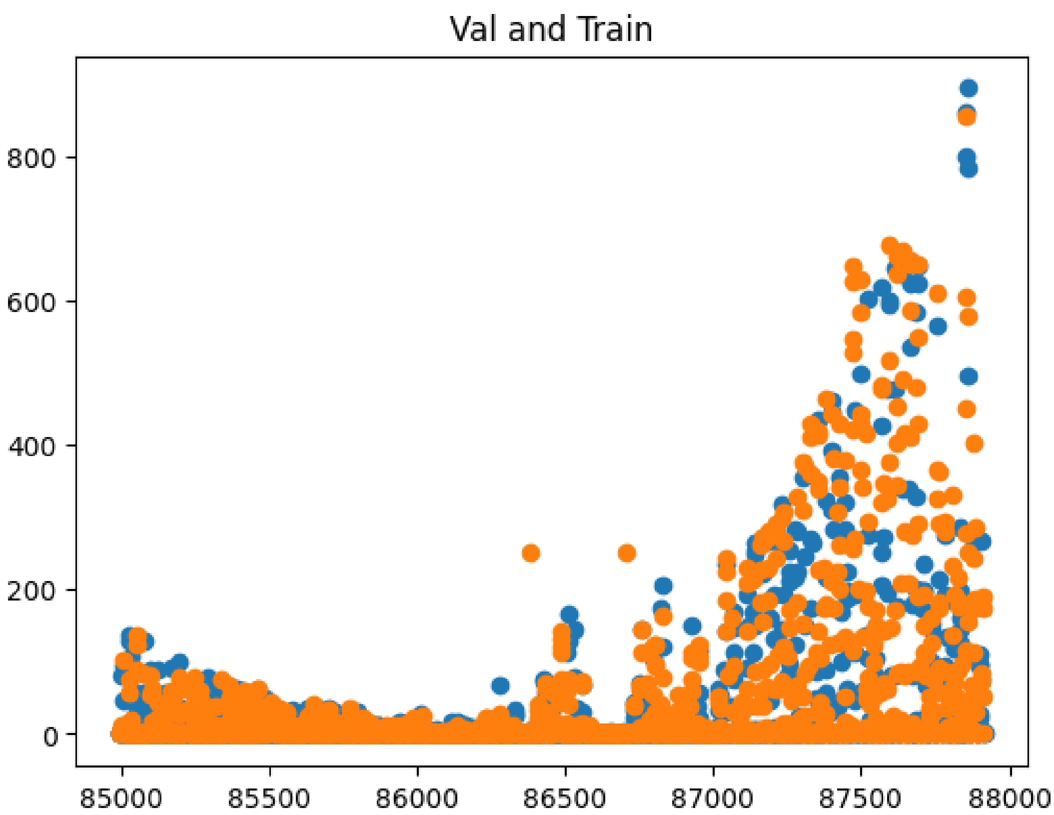
```
leaderboards[2] = leaderboard_for_location(2, loc)
```

```
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=0, num_bag_folds=8,
num_bag_sets=20
Beginning AutoGluon training … Time limit = 600s

Training model for location C…

AutoGluon will save models to "AutogluonModels/submission_120_C/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   167.20 GB / 315.93 GB (52.9%)
Train Data Rows:    24073
Train Data Columns: 44
Tuning Data Rows:    1481
Tuning Data Columns: 44
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and label-values can't be converted to int).
        Label info (max, min, mean, stddev): (999.6, -0.0, 80.92366, 169.71514)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    129972.03 MB
        Train Data (Original)  Memory Usage: 10.27 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 2 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 3): ['elevation:m', 'snow_drift:idx',
'location']
                These features carry no predictive signal and should be manually
```

```
investigated.
		This is typically a feature which has the same value for all
rows.
		These features do not need to be present at inference time.
	Types of features in original data (raw dtype, special dtypes):
		('float', []) : 40 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
		('int', [])   :  1 | ['is_estimated']
	Types of features in processed data (raw dtype, special dtypes):
		('float', [])     : 39 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
		('int', ['bool']) :  2 | ['snow_density:kgm3', 'is_estimated']
	0.1s = Fit runtime
	41 features in original data used to generate 41 features in processed
data.
	Train Data (Processed) Memory Usage: 8.02 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.16s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
	This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
	To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
	'NN_TORCH': {},
	'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
	'CAT': {},
	'XGB': {},
	'FASTAI': {},
	'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
	'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
	'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
```

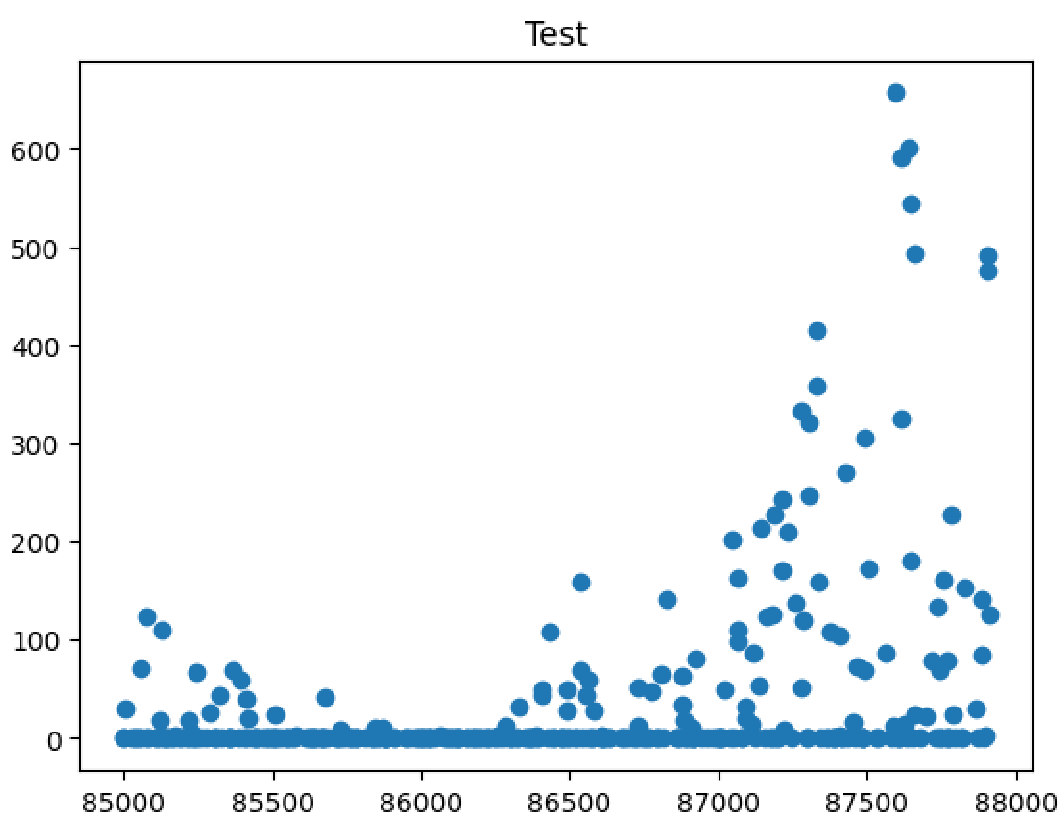
```
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 599.84s of the
599.84s of remaining time.
        -20.8989         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.25s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 599.5s of the
599.49s of remaining time.
        -20.9094         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.25s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 599.15s of the
599.15s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -12.0729         = Validation score   (-mean_absolute_error)
        30.24s   = Training   runtime
        11.44s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 565.03s of the
565.03s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.5515         = Validation score   (-mean_absolute_error)
        31.78s   = Training   runtime
        10.17s   = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 528.71s of
the 528.71s of remaining time.
        -17.3315         = Validation score   (-mean_absolute_error)
        5.69s    = Training   runtime
        0.78s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 521.68s of the
521.67s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.9695         = Validation score   (-mean_absolute_error)
        198.31s  = Training   runtime
        0.08s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 322.14s of the
322.14s of remaining time.
        -16.4795         = Validation score   (-mean_absolute_error)
        1.16s    = Training   runtime
        0.79s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 319.56s of
the 319.56s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.4595         = Validation score   (-mean_absolute_error)
        30.68s   = Training   runtime
        0.41s    = Validation runtime
```

```
Fitting model: XGBoost_BAG_L1 … Training model for up to 287.4s of the 287.4s
of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.8674         = Validation score   (-mean_absolute_error)
        85.54s   = Training   runtime
        7.01s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 197.79s of the
197.79s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.879  = Validation score   (-mean_absolute_error)
        92.57s   = Training   runtime
        0.37s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 103.81s of the
103.81s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.8418         = Validation score   (-mean_absolute_error)
        87.54s   = Training   runtime
        21.08s   = Validation runtime
Completed 1/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
5.79s of remaining time.
        -11.8202         = Validation score   (-mean_absolute_error)
        0.41s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 594.64s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_120_C/")
Evaluation: mean_absolute_error on test data: -10.4514646922694
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -10.4514646922694,
    "root_mean_squared_error": -27.262978710288,
    "mean_squared_error": -743.2700081576166,
    "r2": 0.9206671840046328,
    "pearsonr": 0.9603633994302277,
    "median_absolute_error": -0.5049956440925598
}

Evaluation on test data:
-10.4514646922694
```




```
                     model  score_test  score_val  pred_time_test  pred_time_val
fit_time  pred_time_test_marginal  pred_time_val_marginal  fit_time_marginal
stack_level  can_infer  fit_order
0      WeightedEnsemble_L2  -10.451465 -11.820193        1.323739      12.722344
153.947462                 0.004523                0.000636           0.410090
2        True         12
1    NeuralNetTorch_BAG_L1  -11.174951 -13.878967        0.185156       0.365221
92.565934                 0.185156                0.365221          92.565934
1        True         10
2         LightGBMXT_BAG_L1  -11.372278 -12.072921        0.931536      11.441813
30.240146                 0.931536               11.441813          30.240146
1        True          3
3      LightGBMLarge_BAG_L1  -11.908614 -13.841809        2.644332      21.078984
87.542090                 2.644332               21.078984          87.542090
1        True         11
4    NeuralNetFastAI_BAG_L1  -12.082921 -14.459550        0.172995       0.410839
30.675515                 0.172995                0.410839          30.675515
1        True          8
5           LightGBM_BAG_L1  -12.258116 -13.551534        0.887444      10.165693
31.776026                 0.887444               10.165693          31.776026
1        True          4
```

```
6           CatBoost_BAG_L1  -12.318083 -13.969486        0.071769        0.080049
198.305105                 0.071769                0.080049         198.305105
1       True           6
7      ExtraTreesMSE_BAG_L1  -12.795013 -16.479482        0.272808        0.791115
1.164020                 0.272808                0.791115         1.164020
1       True           7
8            XGBoost_BAG_L1  -12.982353 -14.867444        1.304321        7.007897
85.540600                 1.304321                7.007897         85.540600
1       True           9
9    RandomForestMSE_BAG_L1  -14.231126 -17.331525        0.246395        0.781310
5.687054                 0.246395                0.781310         5.687054
1       True           5
10   KNeighborsUnif_BAG_L1  -17.240469 -20.898903        0.013730        0.252544
0.027626                 0.013730                0.252544         0.027626
1       True           1
11   KNeighborsDist_BAG_L1  -17.539072 -20.909394        0.015799        0.251292
0.028152                 0.015799                0.251292         0.028152
1       True           2
```



[22]:
```
# save leaderboards to csv
pd.concat(leaderboards).to_csv(f"leaderboards/{new_filename}.csv")
```

```
for i in range(len(predictors)):
    print(f"Predictor {i}:")
    print(predictors[i].
    ↪info()["model_info"]["WeightedEnsemble_L2"]["children_info"]["S1F1"]["model_weights"])
```

```
Predictor 0:
{'LightGBMXT_BAG_L1': 0.5913978494623656, 'NeuralNetTorch_BAG_L1':
0.40860215053763443}
Predictor 1:
{'LightGBMXT_BAG_L1': 0.417910447761194, 'ExtraTreesMSE_BAG_L1':
0.029850746268656716, 'NeuralNetFastAI_BAG_L1': 0.07462686567164178,
'NeuralNetTorch_BAG_L1': 0.47761194029850745}
Predictor 2:
{'KNeighborsUnif_BAG_L1': 0.038461538461538464, 'KNeighborsDist_BAG_L1':
0.01282051282051282, 'LightGBMXT_BAG_L1': 0.7051282051282052,
'NeuralNetFastAI_BAG_L1': 0.05128205128205128, 'NeuralNetTorch_BAG_L1':
0.19230769230769232}
```

5 Submit

[23]:
```
import pandas as pd
import matplotlib.pyplot as plt

future_test_data = TabularDataset('X_test_raw.csv')
future_test_data["ds"] = pd.to_datetime(future_test_data["ds"])
#test_data
```

```
Loaded data from: X_test_raw.csv | Columns = 45 / 45 | Rows = 4608 -> 4608
```

[24]:
```
test_ids = TabularDataset('test.csv')
test_ids["time"] = pd.to_datetime(test_ids["time"])
# merge test_data with test_ids
future_test_data_merged = pd.merge(future_test_data, test_ids, how="inner",␣
↪right_on=["time", "location"], left_on=["ds", "location"])

#test_data_merged
```

```
Loaded data from: test.csv | Columns = 4 / 4 | Rows = 2160 -> 2160
```

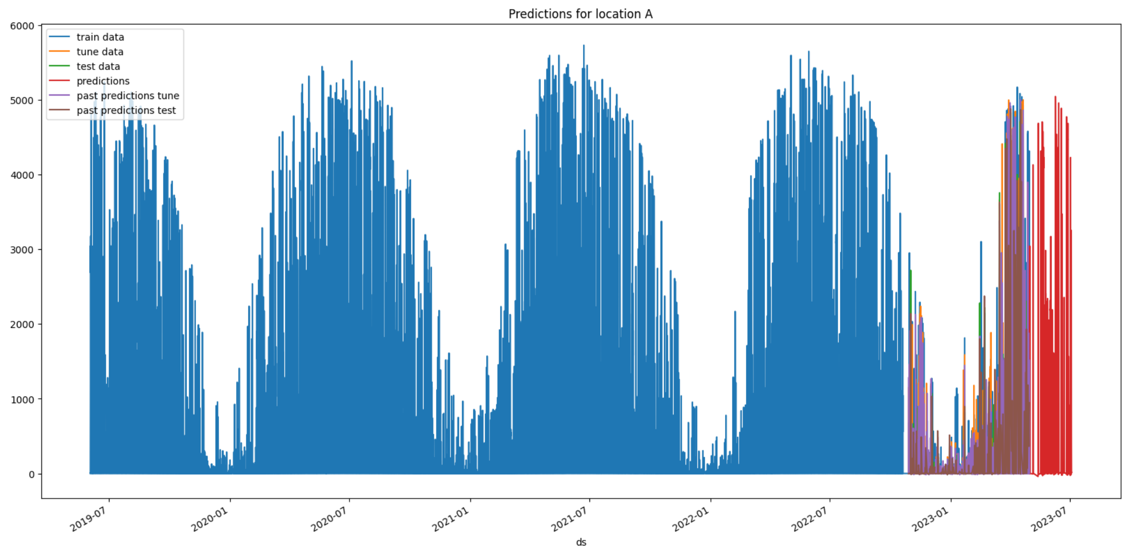
[25]:
```
# predict, grouped by location
predictions = []
location_map = {
    "A": 0,
    "B": 1,
    "C": 2
}
```

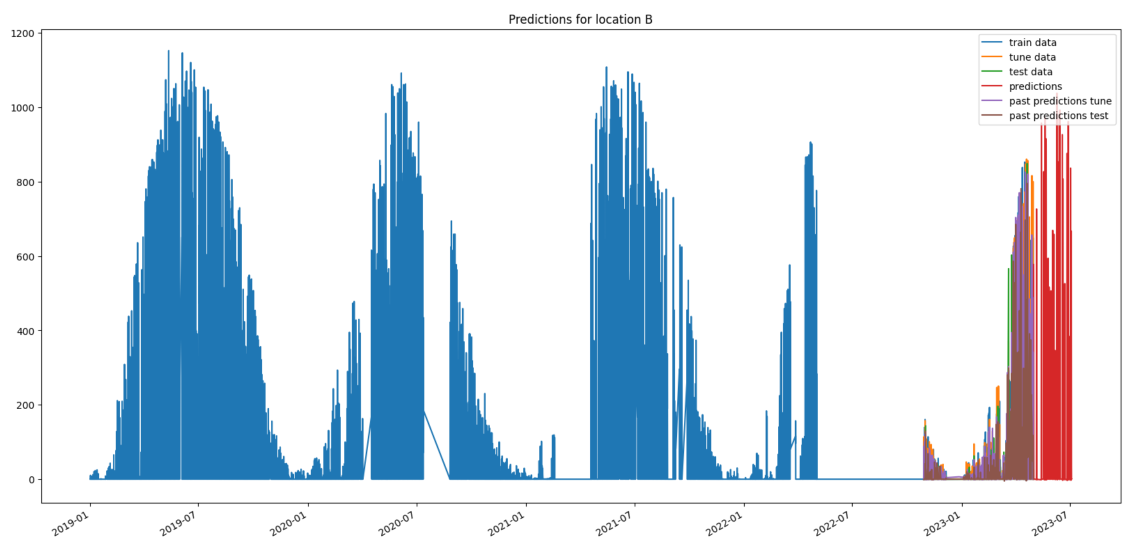
```
for loc, group in future_test_data.groupby('location'):
    i = location_map[loc]
    subset = future_test_data_merged[future_test_data_merged["location"] ==␣
↪loc].reset_index(drop=True)
    #print(subset)
    pred = predictors[i].predict(subset)
    subset["prediction"] = pred
    predictions.append(subset)

    # get past predictions
    #train_data.loc[train_data["location"] == loc, "prediction"] = ␣
↪predictors[i].predict(train_data[train_data["location"] == loc])
    if use_tune_data:
        tuning_data.loc[tuning_data["location"] == loc, "prediction"] = ␣
↪predictors[i].predict(tuning_data[tuning_data["location"] == loc])
    if use_test_data:
        test_data.loc[test_data["location"] == loc, "prediction"] = ␣
↪predictors[i].predict(test_data[test_data["location"] == loc])
```

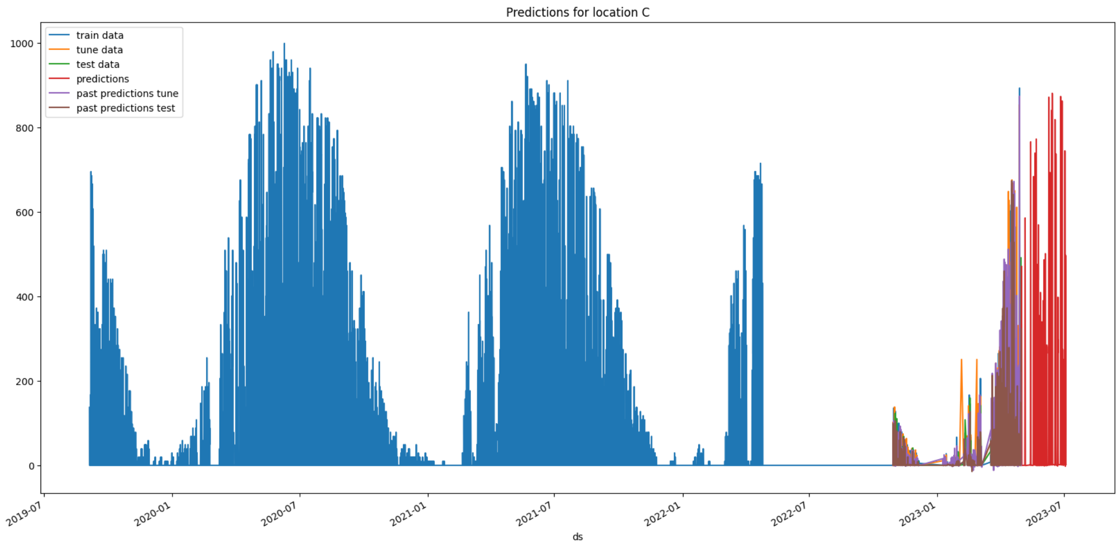
[26]:

```
# plot predictions for location A, in addition to train data for A
for loc, idx in location_map.items():
    fig, ax = plt.subplots(figsize=(20, 10))
    # plot train data
    train_data[train_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
↪label="train data")
    if use_tune_data:
        tuning_data[tuning_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
↪label="tune data")
    if use_test_data:
        test_data[test_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
↪label="test data")

    # plot predictions
    predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")

    # plot past predictions
    #train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
↪y='prediction', ax=ax, label="past predictions")
    #train_data[train_data["location"]==loc].plot(x='ds', y='prediction',␣
↪ax=ax, label="past predictions train")
    if use_tune_data:
        tuning_data[tuning_data["location"]==loc].plot(x='ds', y='prediction',␣
↪ax=ax, label="past predictions tune")
    if use_test_data:
        test_data[test_data["location"]==loc].plot(x='ds', y='prediction',␣
↪ax=ax, label="past predictions test")
```

```python
# title
ax.set_title(f"Predictions for location {loc}")
```







[27]:

```
temp_predictions = [prediction.copy() for prediction in predictions]
if clip_predictions:
    # clip predictions smaller than 0 to 0
    for pred in temp_predictions:
        # print smallest prediction
        print("Smallest prediction:", pred["prediction"].min())
        pred.loc[pred["prediction"] < 0, "prediction"] = 0
        print("Smallest prediction after clipping:", pred["prediction"].min())

# Instead of clipping, shift all prediction values up by the largest negative␣
↪number.
# This way, the smallest prediction will be 0.
elif shift_predictions:
    for pred in temp_predictions:
        # print smallest prediction
        print("Smallest prediction:", pred["prediction"].min())
        pred["prediction"] = pred["prediction"] - pred["prediction"].min()
        print("Smallest prediction after clipping:", pred["prediction"].min())

elif shift_predictions_by_average_of_negatives_then_clip:
    for pred in temp_predictions:
        # print smallest prediction
        print("Smallest prediction:", pred["prediction"].min())
        mean_negative = pred[pred["prediction"] < 0]["prediction"].mean()
        # if not nan
        if mean_negative == mean_negative:
            pred["prediction"] = pred["prediction"] - mean_negative
```

```
        pred.loc[pred["prediction"] < 0, "prediction"] = 0
        print("Smallest prediction after clipping:", pred["prediction"].min())



# concatenate predictions
submissions_df = pd.concat(temp_predictions)
submissions_df = submissions_df[["id", "prediction"]]
submissions_df
```

```
Smallest prediction: -40.46703
Smallest prediction after clipping: 0.0
Smallest prediction: -3.1742952
Smallest prediction after clipping: 0.0
Smallest prediction: -2.212441
Smallest prediction after clipping: 0.0
```

[27]:

```
       id  prediction
0       0    0.351390
1       1    0.238262
2       2    0.000000
3       3   26.153254
4       4  281.920868
..    ...         ...
715  2155   66.743126
716  2156   37.417755
717  2157   10.034138
718  2158    1.761877
719  2159    1.615402

[2160 rows x 2 columns]
```

[28]:

```
# Save the submission DataFrame to submissions folder, create new name based on␣
 ↪last submission, format is submission_<last_submission_number + 1>.csv

# Save the submission
print(f"Saving submission to submissions/{new_filename}.csv")
submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"),␣
 ↪index=False)
print("jall1a")
```

```
Saving submission to submissions/submission_120.csv
jall1a
```

[ ]:

```
# feature importance
# print starting calculating feature importance for location A with big text␣
 ↪font
```

```
print("\033[1m" + "Calculating feature importance for location A..." +␣
 ↪"\033[0m")
predictors[0].feature_importance(feature_stage="original",␣
 ↪data=test_data[test_data["location"] == "A"], time_limit=60*10)
print("\033[1m" + "Calculating feature importance for location B..." +␣
 ↪"\033[0m")
predictors[1].feature_importance(feature_stage="original",␣
 ↪data=test_data[test_data["location"] == "B"], time_limit=60*10)
print("\033[1m" + "Calculating feature importance for location C..." +␣
 ↪"\033[0m")
predictors[2].feature_importance(feature_stage="original",␣
 ↪data=test_data[test_data["location"] == "C"], time_limit=60*10)
```

```
These features in provided data are not utilized by the predictor and will be
ignored: ['ds', 'elevation:m', 'snow_drift:idx', 'location', 'prediction']
Computing feature importance via permutation shuffling for 41 features using 361
rows with 10 shuffle sets… Time limit: 600s…
```

Calculating feature importance for location A…

```
        476.61s = Expected runtime (47.66s per shuffle set)
        58.85s  = Actual runtime (Completed 10 of 10 shuffle sets)
These features in provided data are not utilized by the predictor and will be
ignored: ['ds', 'elevation:m', 'location', 'prediction']
Computing feature importance via permutation shuffling for 42 features using 361
rows with 10 shuffle sets… Time limit: 600s…
```

Calculating feature importance for location B…

```
        869.28s = Expected runtime (86.93s per shuffle set)
        86.31s  = Actual runtime (Completed 10 of 10 shuffle sets)
These features in provided data are not utilized by the predictor and will be
ignored: ['ds', 'elevation:m', 'snow_drift:idx', 'location', 'prediction']
Computing feature importance via permutation shuffling for 41 features using 360
rows with 10 shuffle sets… Time limit: 600s…
```

Calculating feature importance for location C…

```
        637.25s = Expected runtime (63.72s per shuffle set)
```

[ ]:
```
# save this notebook to submissions folder
import subprocess
import os
#subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
 ↪join('notebook_pdfs', f"{new_filename}_automatic_save.pdf"),␣
 ↪"autogluon_each_location.ipynb"])
subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
 ↪join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.
 ↪ipynb"])
```

[ ]:

```
# import subprocess

# def execute_git_command(directory, command):
#     """Execute a Git command in the specified directory."""
#     try:
#         result = subprocess.check_output(['git', '-C', directory] + command,␣
↪stderr=subprocess.STDOUT)
#         return result.decode('utf-8').strip(), True
#     except subprocess.CalledProcessError as e:
#         print(f"Git command failed with message: {e.output.decode('utf-8').
↪strip()}")
#         return e.output.decode('utf-8').strip(), False

# git_repo_path = "."

# execute_git_command(git_repo_path, ['config', 'user.email',␣
↪'henrikskog01@gmail.com'])
# execute_git_command(git_repo_path, ['config', 'user.name', hello if hello is␣
↪not None else 'Henrik eller Jørgen'])

# branch_name = new_filename

# # add datetime to branch name
# branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"

# commit_msg = "run result"

# execute_git_command(git_repo_path, ['checkout', '-b',branch_name])

# # Navigate to your repo and commit changes
# execute_git_command(git_repo_path, ['add', '.'])
# execute_git_command(git_repo_path, ['commit', '-m',commit_msg])

# # Push to remote
# output, success = execute_git_command(git_repo_path, ['push',␣
↪'origin',branch_name])

# # If the push fails, try setting an upstream branch and push again
# if not success and 'upstream' in output:
#     print("Attempting to set upstream and push again...")
#     execute_git_command(git_repo_path, ['push', '--set-upstream',␣
↪'origin',branch_name])
#     execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])

# execute_git_command(git_repo_path, ['checkout', 'main'])
```

[ ]:

```
```